Guida Pratica: Analisi e Sfruttamento SQL Injection su DVWA

Autore: Kevin Motti

Laboratorio: Damn Vulnerable Web App (DVWA)

Obiettivo: Dimostrazione pratica di esfiltrazione dati e recupero credenziali.

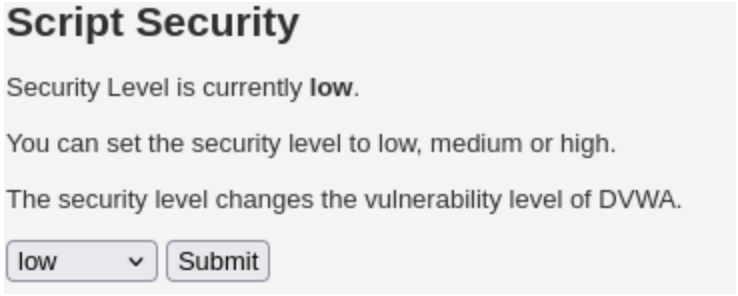
1. Introduzione: Cos'è successo?

In questo laboratorio abbiamo testato la sicurezza di un'applicazione web volutamente vulnerabile (DVWA). L'obiettivo era ingannare il database del sito per fargli rivelare informazioni riservate (nomi utenti e password) che non avremmo dovuto vedere.

Abbiamo eseguito l'attacco su due livelli di difficoltà: **Low** (Basso) e **Medium** (Medio), per vedere come cambiano le difese e come aggirarle.

2. Livello di Sicurezza: LOW (Basso)

In questo livello, il sito non ha protezioni. Prende esattamente quello che scriviamo e lo invia al database.



Passo 1: Trovare il punto debole

L'applicazione ci chiede di inserire un **User ID** (un numero) per vedere il nome di un utente.

- **Comportamento normale:** Se scrivo 1, il sito cerca l'utente con ID 1.
- **Il problema:** Il sito non controlla se inserisco *solo* un numero.

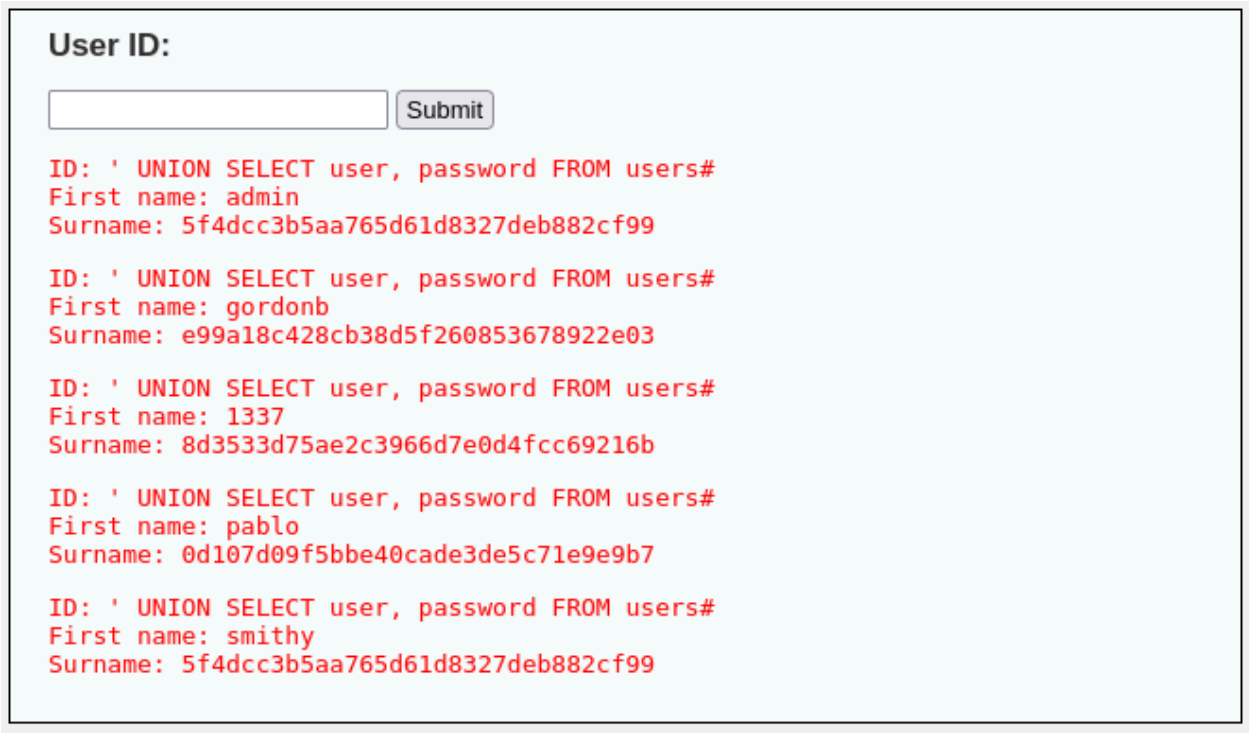
Passo 2: L'Attacco (La "Query")

Abbiamo inserito un comando speciale invece del numero.

- **Il Payload (il codice malevolo):** 'UNION SELECT user, password FROM users #
- **Cosa significa:**

1. ' : Chiude la ricerca normale del sito.
2. UNION : Unisce i risultati della ricerca normale con la *nostra* nuova ricerca.
3. SELECT user, password FROM users : Chiede al database di darci la lista di utenti e password.
4. # : Dice al database di ignorare (commentare) tutto quello che c'è dopo, per evitare errori.

Risultato



Il sito ha mostrato a schermo non solo il nome dell'utente cercato, ma **l'intera lista degli utenti e le loro password** (sotto forma di codici hash MD5).

3. Livello di Sicurezza: MEDIUM (Medio)

Qui le cose si complicano. Il sito ha attivato due difese:

1. **Filtro caratteri:** Blocca l'uso dell'apice singolo ('), quindi il trucco usato prima non funziona.
2. **Menu a tendina:** Non possiamo più scrivere liberamente; dobbiamo scegliere un numero da un menu predefinito.

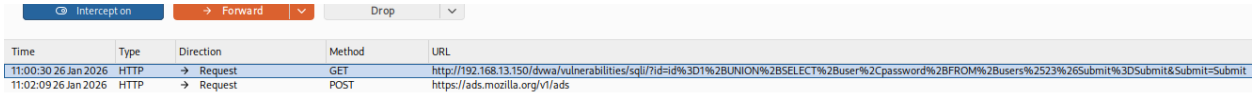
Per superare queste difese, abbiamo usato due metodi diversi.

Metodo A: Intercettare la richiesta tramite Burp Suite (proxy)

Siccome il menu a tendina ci impedisce di scrivere, abbiamo intercettato la comunicazione tra il nostro browser e il server.

Procedura: 1.

1. Selezionato un ID valido dall'interfaccia e cliccato "Submit".
2. Intercettata la richiesta POST (o GET) su Burp Suite.



3. Inviata la richiesta al modulo Repeater.
4. Modificato il parametro id inserendo il payload senza apici.

```
GET /dvwa/vulnerabilities/sqli/?id=1+UNION+SELECT+user,password+FROM+users%23&Submit=Submit
HTTP/1.1
```

Payload Utilizzato (URL Encoded):

1+UNION+SELECT+user,password+FROM+users%23

Risultato:

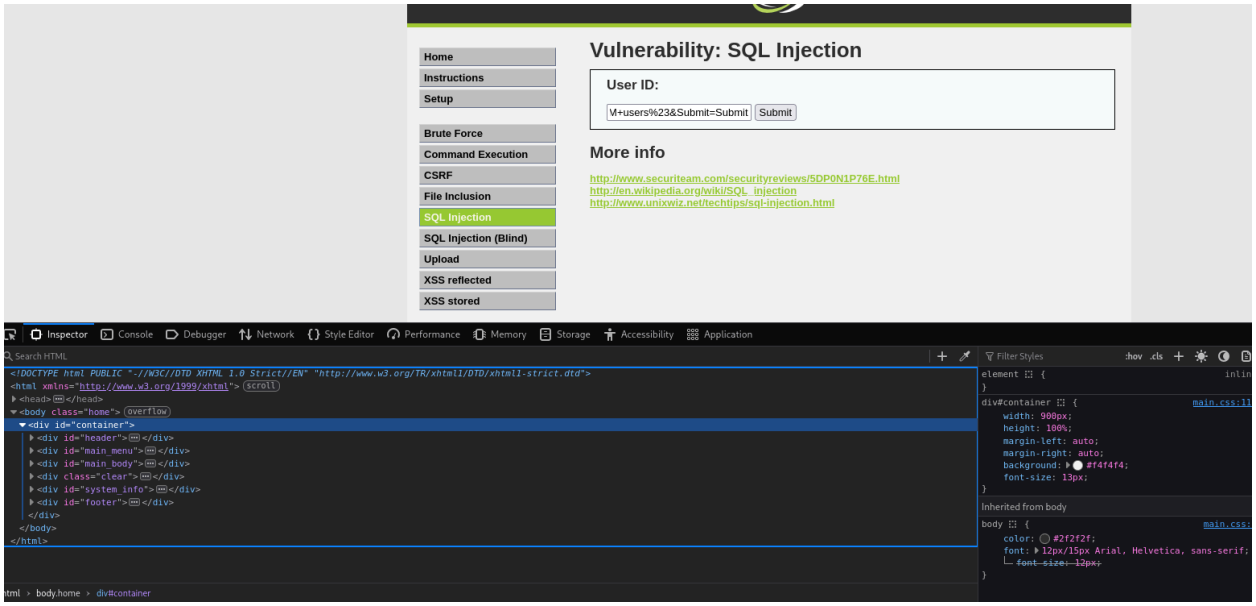
```
<pre>
     ID: 1 UNION SELECT user,password FROM users#<br>
     First name: admin<br>
     Surname: admin
</pre>
<pre>
     ID: 1 UNION SELECT user,password FROM users#<br>
     First name: admin<br>
     Surname: 5f4dcc3b5aa765d61d8327deb882cf99
</pre>
<pre>
     ID: 1 UNION SELECT user,password FROM users#<br>
     First name: gordonb<br>
     Surname: e99a18c428cb38d5f260853678922e03
</pre>
<pre>
     ID: 1 UNION SELECT user,password FROM users#<br>
     First name: 1337<br>
     Surname: 8d3533d75ae2c3966d7e0d4fcc69216b
</pre>
<pre>
     ID: 1 UNION SELECT user,password FROM users#<br>
     First name: pablo<br>
     Surname: 0d107d09f5bbe40cade3de5c71e9e9b7
</pre>
<pre>
     ID: 1 UNION SELECT user,password FROM users#<br>
     First name: smithy<br>
     Surname: 5f4dcc3b5aa765d61d8327deb882cf99
</pre>
```

Il server ha eseguito la query iniettata restituendo nuovamente gli hash delle password nella risposta HTTP (pannello Response di Burp).

Metodo B: Client-Side Bypass (HTML Inspector)

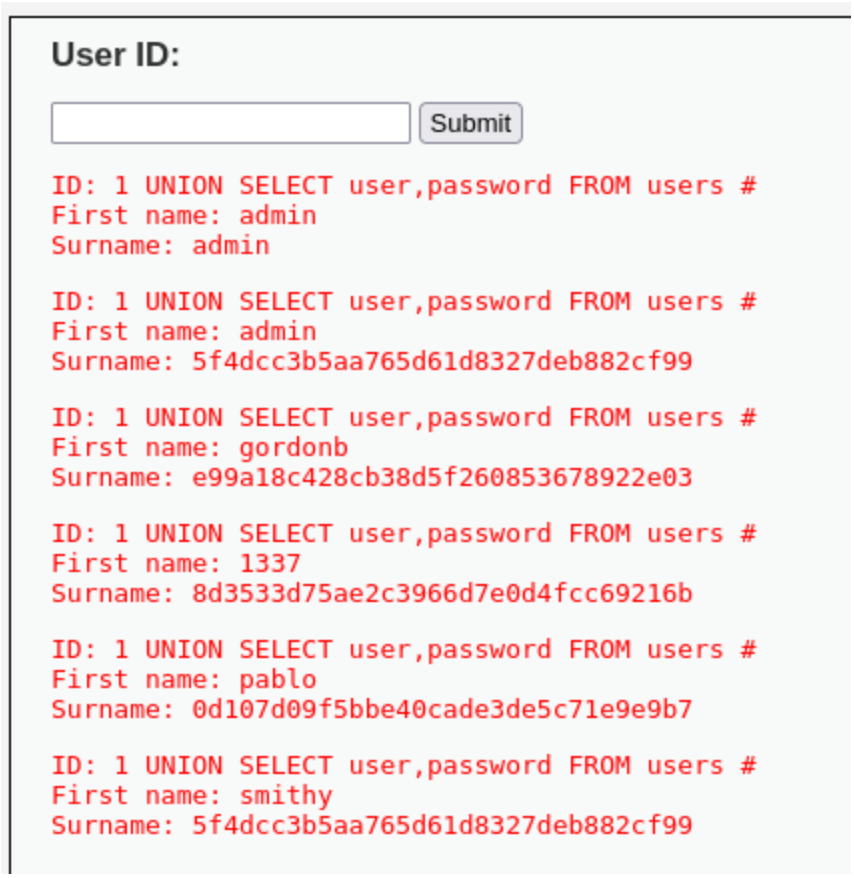
Le protezioni grafiche (come i menu a tendina) sono facili da aggirare perché vivono sul *nostro* browser.

1. **Ispeziona:** Abbiamo cliccato col tasto destro sul menu a tendina e scelto "Ispeziona".



2. **La Modifica:** È stato modificato l'attributo value dell'elemento HTML, sostituendo l'ID numerico legittimo con il payload SQL malevolo.
    - Payload inserito nel DOM HTML: 1 UNION SELECT user, password FROM users #

3. **L'Invio:** Cliccando "Submit", il browser ha inviato il nostro codice modificato invece del numero originale.



Risultato: Anche in questo caso, il database ha risposto stampando a schermo tutti gli utenti e le password criptate.

4. Post-Exploitation: Recupero Password in Chiaro

Le password estratte dal database (es. campo surname) non erano in chiaro, ma cifrate con algoritmo MD5. Come richiesto dalla traccia, è stato necessario un "ulteriore step" per renderle leggibili.

Dati Estratti (Esempio):

- Utente: pablo
- Hash MD5: 0d107d09f5bbe40cade3de5c71e9e9b7

Procedura di Cracking

Per leggere la password in chiaro dell'utente pablo, abbiamo usato uno strumento chiamato **John the Ripper**.

1. Con il comando **mousepad pablo_picasso.txt** abbiamo salvato l'hash in un file di testo.

```
1 pablo picasso:0d107d09f5bbe40cade3de5c71e9e9b7
```

2. Abbiamo lanciato *John the Ripper* dicendogli che il formato era Raw-MD5.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ john --show --format=RAW-MD5 pablo_picasso.txt
pablo picasso:letmein
```

3. Il programma ha confrontato l'hash con milioni di parole comuni fino a trovare quella che corrispondeva.

Risultato:

- **Hash:** 0d107d09f5bbe40cade3de5c71e9e9b7
- **Password scoperta:** letmein

L'accesso all'account dell'utente pablo è stato verificato con successo utilizzando questa password:

> Welcome to the password protected area pablo

5. Conclusioni

L'esercitazione ha dimostrato che la sanificazione parziale degli input (come nel livello Medium) e le restrizioni dell'interfaccia grafica non sono sufficienti a proteggere un database. Un attaccante può manipolare le richieste HTTP o il codice HTML locale per iniettare comandi SQL arbitrari se il backend non utilizza non utilizza Prepared Statements parametrizzati.